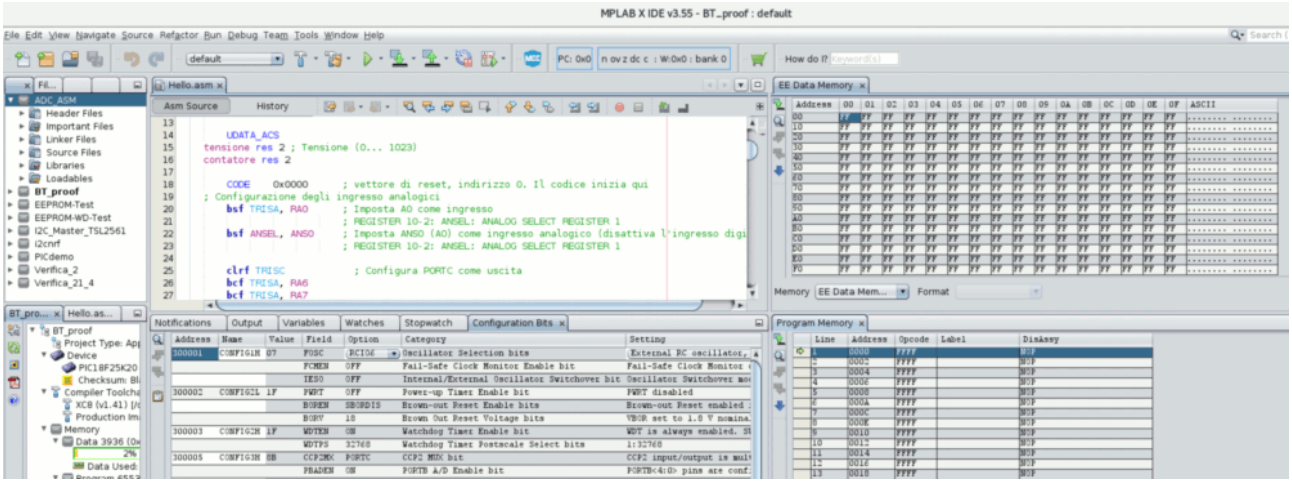
PIC18 e assembly: il primo programma



In questa pagina vedremo come creare il primo progetto e come usare il simulatore per esplorare la struttura ed il contenuto delle memorie del microcontrollore che stiamo utilizzando. Non faremo uso di circuiti fisici perché è cosa complessa capire se un comportamento imprevisto nasce da un errore hardware oppure software.

La creazione di un nuovo progetto

Ciascun programma deve essere incluso in un progetto. La procedura per creare un progetto è guidata:

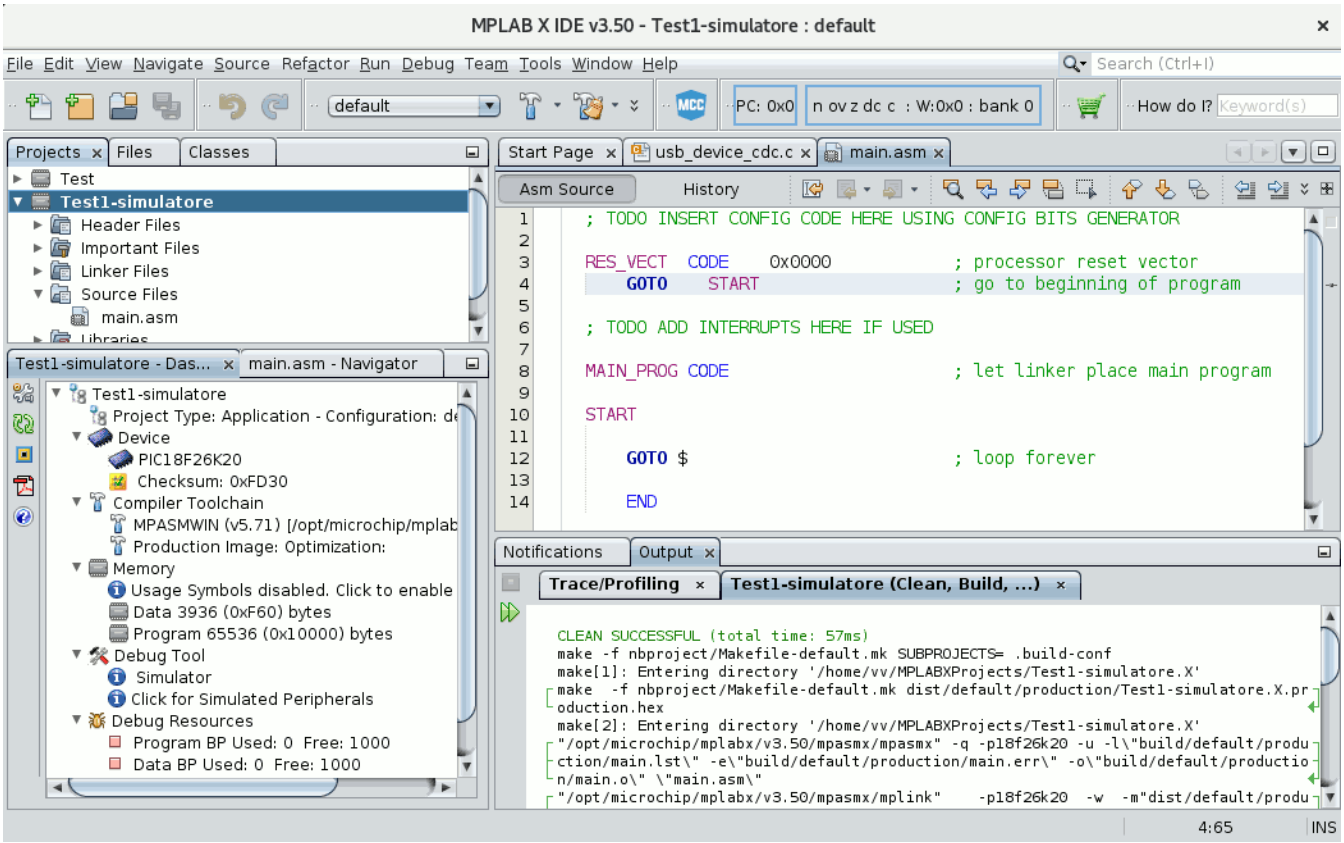
1. Inizia ovviamente con *Create New Project*. Le scelte di default (*Microchip embedded* e *Standalone Project*) sono quelle in genere corrette
2. Occorre quindi scegliere la variante di PIC18 (è facile sbagliare: molte sigle si assomigliano). L'esempio mostrato in questa pagina utilizza un PIC18F45K22, ma nulla cambia scegliendo un altro PIC18
3. Se intendiamo utilizzare circuiti fisici (ma non è il caso qui mostrato) viene richiesto il modello di *In-Circuit Debugger*. In questo primo esempio sceglieremo invece il simulatore (*Simulator*)
4. L'assemblatore da utilizzare (in seguito MPASM). Se sono presenti più versioni, potrebbe essere conveniente scegliere quella più aggiornata
5. Il nome del progetto

Quando si crea un nuovo programma è necessario creare un nuovo progetto oppure fare una copia del vecchio progetto (tasto destro del mouse sul nome del progetto)

L'immagine seguente mostra le finestre principali di MPLAB X:

- In alto a sinistra, l'elenco dei progetti. Se sono presenti più progetti, quello in uso (*main project*, in grassetto) deve essere selezionato con il tasto destro.
- Nella stessa finestra: l'elenco dei file del progetto (nell'esempio uno solo: `main.asm`). Progetti semplici sono costituiti da un solo file, progetti più complessi comprendono molti file.
- In alto a destra, il contenuto di un file sorgente (nell'esempio `main.asm`)
- In basso a sinistra, la *dashboard*, che permette di monitorare l'uso delle risorse nonché di modificare le scelte fatte (per esempio: il tipo di processore)
- In basso a destra, i messaggi nelle varie fasi di compilazione, programmazione ed esecuzione. Eventuali errori (qui non presenti) sono evidenziati in rosso
- In alto le varie icone e menu di compilazione e debug

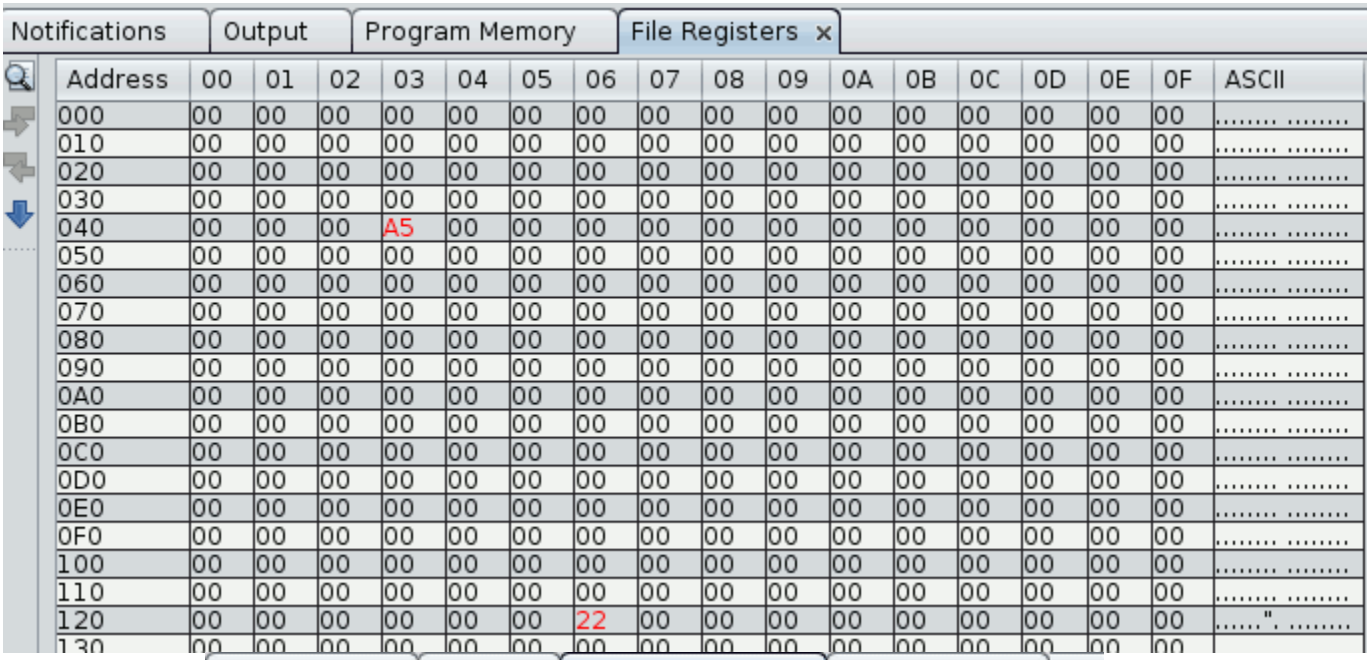
Durante l'uso vengono aperte anche numerose altre finestre...



La memoria

Attraverso il simulatore è possibile esaminare lo stato del microcontrollore. Per esempio, attraverso la voce **Window → PIC Memory view**, è possibile esaminare il contenuto:

- dei *File Registers*, la memoria a lettura/scrittura (RAM) utilizzata per contenere le variabili. Nell'esempio qui sotto mostrato è piena di zeri, tranne le due celle con indirizzo 0x043 e 0x126, evidenziate in rosso
- della *Program Memory*, la memoria a sola lettura (Flash) che contiene il codice eseguibile. In questo caso è piena di 1, tranne le prime celle

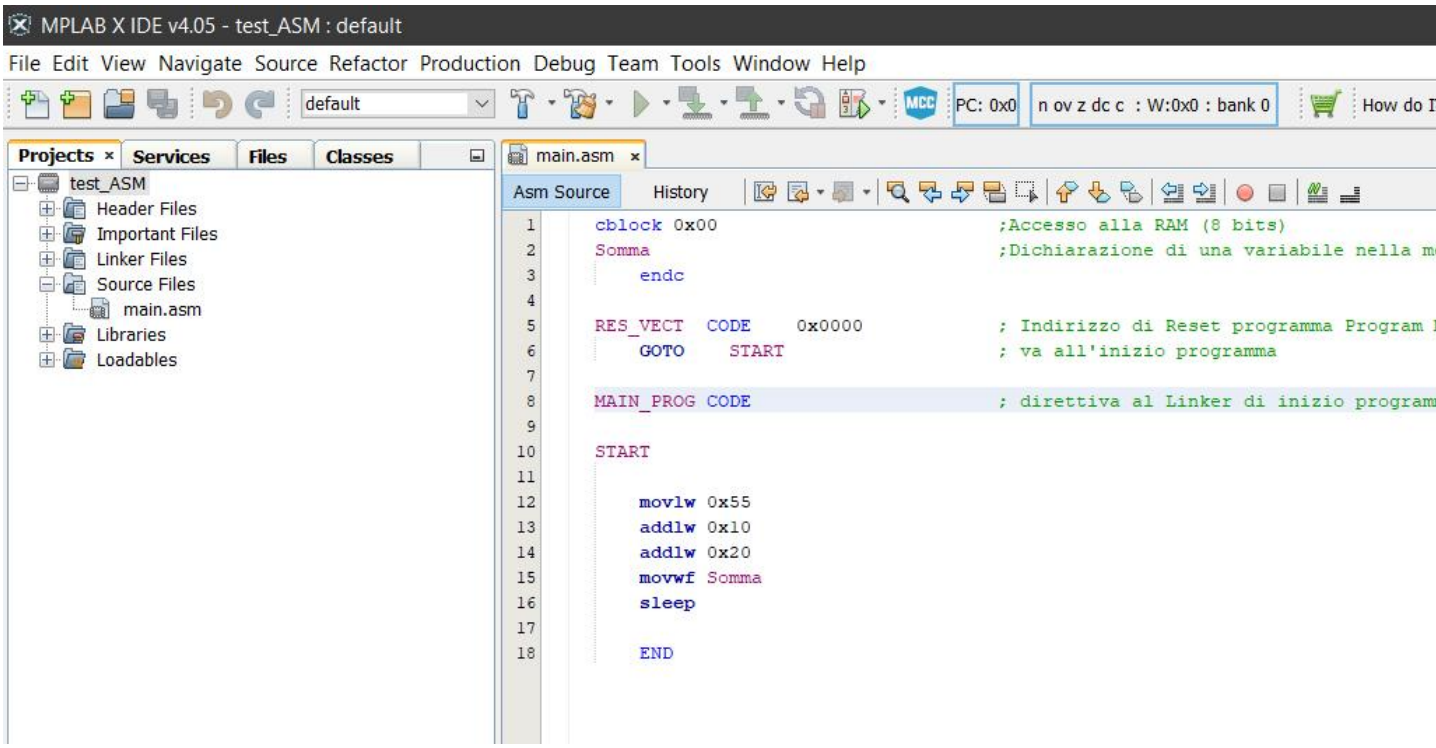


Si noti che queste due memorie hanno una struttura diversa:

- La prima è una memoria RAM volatile, suddivisa a seconda della versione del PIC, in circa 4000 celle da 8 bit ciascuna
- La seconda è una memoria flash non volatile, suddivisa a seconda della versione del PIC, in circa qualche decina di migliaia di celle da 16 bit ciascuna

Il primo programma

Scriviamo il primo programma, creando un file con estensione `.asm`. e salvandolo nella cartella virtuale `Source Files`. Quanto mostrato non fa assolutamente nulla di utile, ma ci servirà per prendere confidenza con il microcontrollore, l'ambiente di sviluppo.



Cosa fa questo programma?

1. Istruzione `movlw 0x55`: **MOV**e **L**etteral to **W** register - muove letteralmente nel registro **W** il numero esadecimale `0x55`.
2. Istruzione `addlw 0x10`: **ADD L**etteral to **W** register - somma letteralmente `0x10` al contenuto del registro **W**, memorizzando il risultato nel registro **W**
3. Istruzione `addlw 0x20`: **ADD L**etteral to **W** register - somma letteralmente `0x20` al contenuto del registro **W**, memorizzando il risultato nel registro in **W**

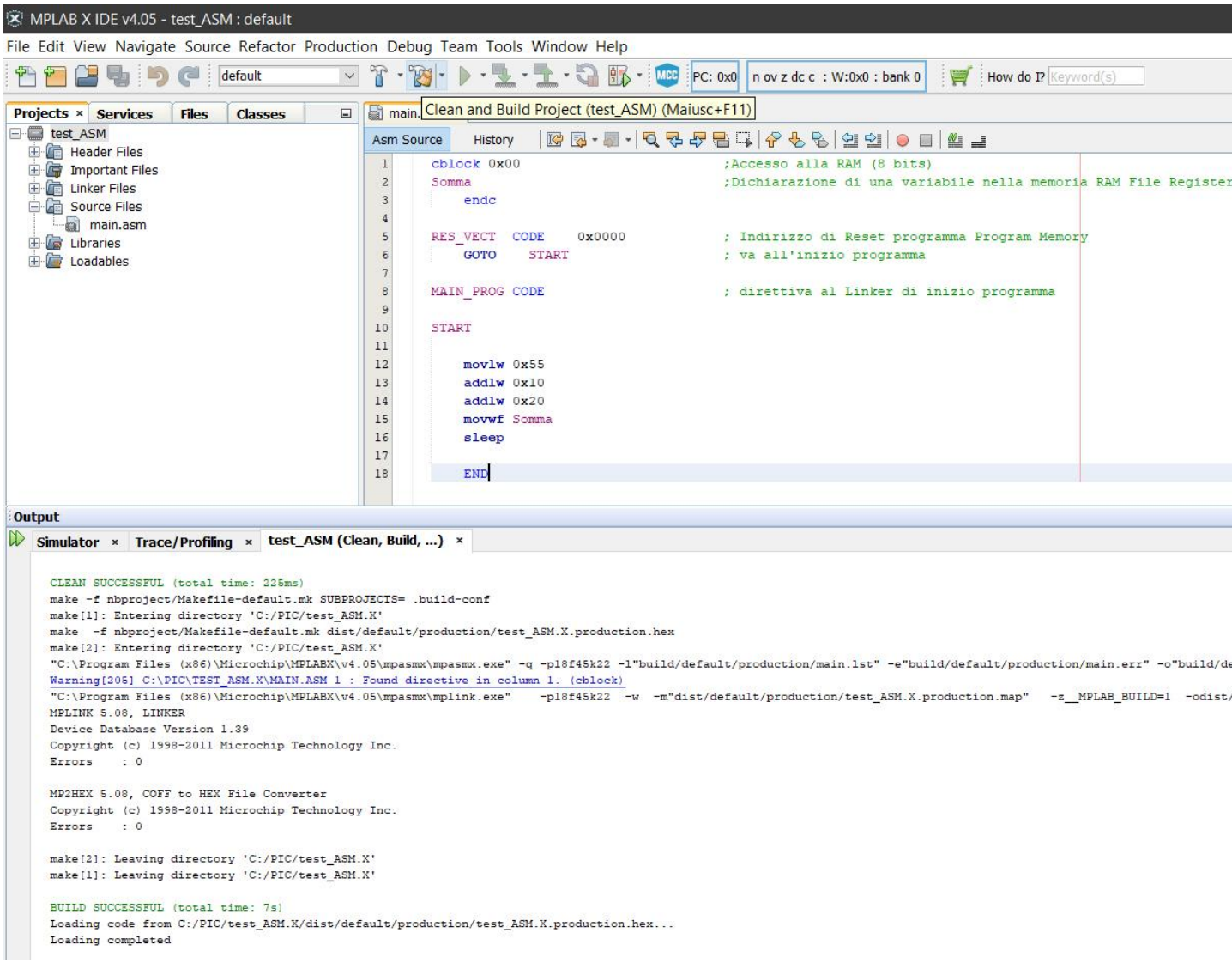
4. Istruzione `movwf Somma`: **sposta** il contenuto dell'Accumulatore W nella Variabile **'Somma'**
5. Istruzione `sleep`: **SLEEP** - addormenta il processore, cioè ne sospende l'attività

Alcune osservazioni:

- è necessario che ciascuna riga di codice abbia almeno uno spazio davanti
- le istruzioni possono essere scritte maiuscole o minuscole, ma in seguito saranno sempre in minuscolo per chiarezza
- la prima e l'ultima riga, in un colore diverso, NON contengono *istruzioni*, ma *direttive* Nello specifico indicano che il codice dovrà essere memorizzato a partire dall'indirizzo di memoria 0 (**Cblock 0x00**) e che dopo l'**END** non c'è più nulla di significativo
- i numeri esadecimali sono indicati con la sintassi tipica del C, ma, in questo caso, non è strettamente necessario, dato che la base 16 è quella predefinita. Quindi, a meno di impostazioni particolari, scrivere `0x10` è la stessa cosa che scrivere `10`. Consiglio vivamente la prima codifica (`0x10`) in quanto la seconda (`10`) può facilmente portare ad errori di interpretazione da parte di chi legge il codice

Assembliamo il programma

Per assemblare il programma appena scritto occorre utilizzare l'icona evidenziata, osservando nella finestra di output l'assenza di errori.



Il programma compilato viene automaticamente caricato nella *Program Memory* del simulatore. Nella figura seguente possiamo osservare

- le cinque istruzioni che abbiamo scritto, ma non le due direttive (colonna *DisAssy*, traducibile con Disassemblato)
- l'indirizzo (*Address*) delle celle di memoria in cui sono state memorizzate (solo indirizzi pari, essendo tutte lunghe 16 bit, cioè quatto cifre esadecimali)
- Il codice binario corrispondente (*Opcode*, codice operativo, le sole istruzioni che un processore può elaborare)

Program Memory

Line	Address	Opcode	Label	DisAssy
1	0000	EF03		GOTO 0x6
2	0002	F000		NOP
3	0004	0000		NOP
4	0006	0E55		MOVLW 0x55
5	0008	0F10		ADDLW 0x10
6	000A	0F20		ADDLW 0x20
7	000C	6E00		MOVWF 0x0, ACCESS
8	000E	0003		SLEEP
9	0010	FFFF		NOP
10	0012	FFFF		NOP
11	0014	FFFF		NOP
12	0016	FFFF		NOP
13	0018	FFFF		NOP

Eseguiamo il programma

L'avvio del programma è gestito da una serie di icone la cui descrizione è mostrata scorrendoci sopra il mouse:

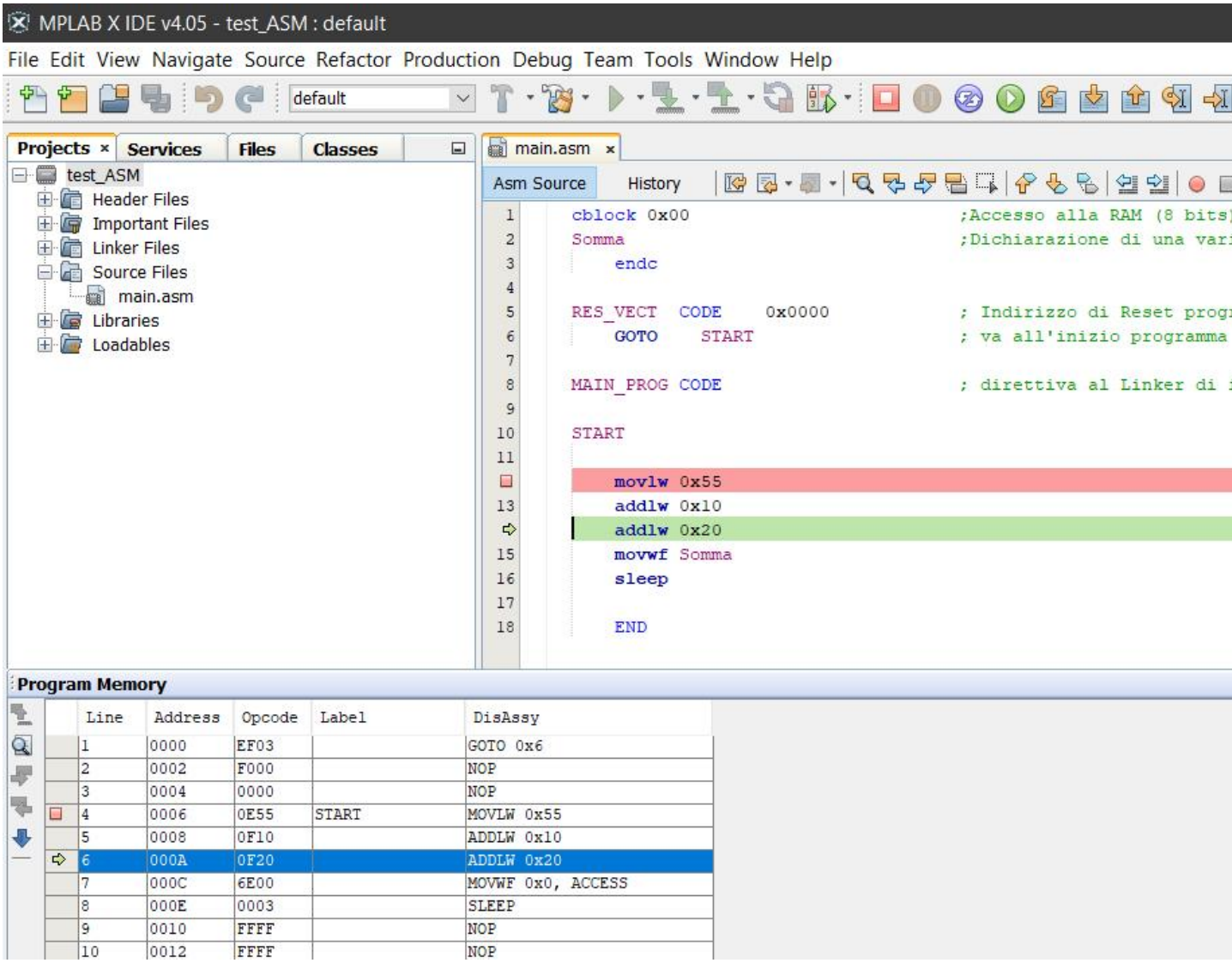
Nell'ordine:

- Inizia l'esecuzione del programma
- Ferma definitivamente l'esecuzione del programma
- (in grigio) sospende temporaneamente l'esecuzione del programma
- Resetta il programma (ricomincia dall'indirizzo zero)
- Prosegue l'esecuzione del programma dal punto in cui è stato precedentemente sospeso
- Esegue una singola istruzione (esecuzione passo-passo). In realtà sono una serie di icone simili, dal comportamento un po' diverso (ma intuitivo...)

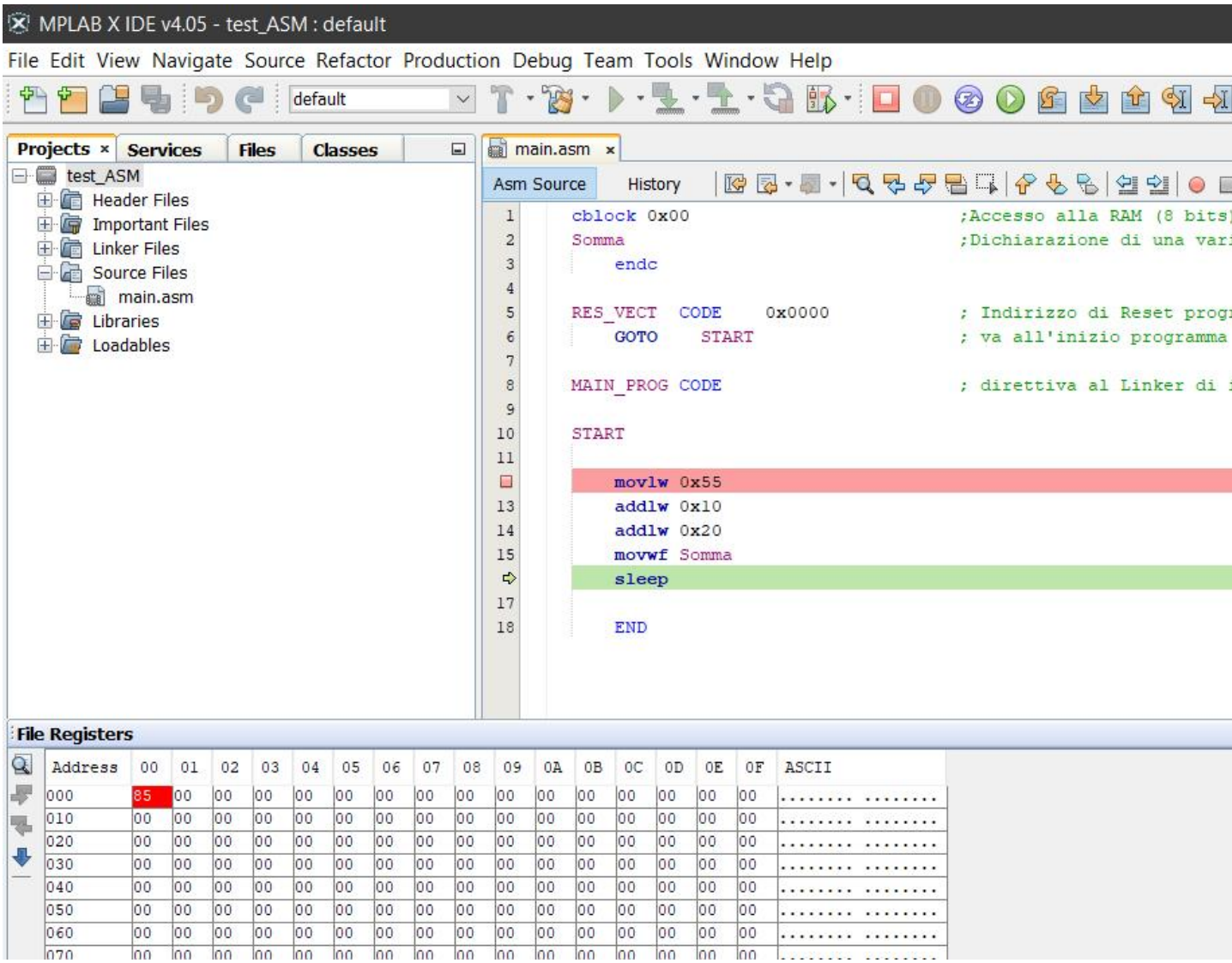
Se si desidera fermare il programma ad una certa istruzione, è possibile inserire un *breakpoint* (linea rossa nell'immagine seguente), cliccando sul corrispondente numero di riga.

Durante l'esecuzione passo-passo possiamo osservare:

- In alto a destra il contenuto del Program Counter (**PC**) e del registro **W.** Inoltre, anche se al momento non serve comprenderne il significato, è segnalato il valore dei flags (o *Status Bits*) e quale è il banco di memoria in uso
- Nella finestra del codice, evidenziato in rosso, un breakpont ed evidenziata in verde e da una freccia l'istruzione che sta per essere eseguita
- Nella *Program Memory*, la freccia verde indica l'istruzione che sta per essere eseguita



Inoltre è possibile osservare e modificare il contenuto della memoria RAM (*File Registers*).


MPLAB X IDE v4.05 - test_ASM : default
File Edit View Navigate Source Refactor Production Debug Team Tools Window Help
default
Projects
Services
Files
Classes
test_ASM
Header Files
Important Files
Linker Files
Source Files
main.asm
Libraries
Loadables
main.asm
Asm Source
History
1 cblock 0x00 ;Accesso alla RAM (8 bits)
2 Somma ;Dichiarazione di una vari
3 endc
4
5 RES_VECT CODE 0x0000 ; Indirizzo di Reset progr
6 GOTO START ; va all'inizio programma
7
8 MAIN_PROG CODE ; direttiva al Linker di
9
10 START
11
13 movlw 0x55
14 addlw 0x10
15 addlw 0x20
17 movwf Somma
18 sleep
END
File Registers
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
000 85 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....